Online Bandwidth Allocation

Michal Forišek Branislav Katreniak Jana Katreniaková
Rastislav Královič Richard Královič Vladimír Koutný
Dana Pardubská Tomáš Plachetka
Branislav Rovan

Dept. of Computer Science, Comenius University,
Mlynská dolina, 84248 Bratislava, Slovakia.

February 7, 2008

Abstract

The paper investigates a version of the resource allocation problem arising in the wireless networking, namely in the OVSF code reallocation process. In this setting a complete binary tree of a given height n is considered, together with a sequence of requests which have to be served in an online manner. The requests are of two types: an insertion request requires to allocate a complete subtree of a given height, and a deletion request frees a given allocated subtree. In order to serve an insertion request it might be necessary to move some already allocated subtrees to other locations in order to free a large enough subtree. We are interested in the worst case average number of such reallocations needed to serve a request.

In [4] the authors delivered bounds on the competitive ratio of online algorithm solving this problem, and showed that the ratio is between 1.5 and $O(n)$. We partially answer their question about the exact value by giving an $O(1)$-competitive online algorithm.

In [3], authors use the same model in the context of memory management systems, and analyze the number of reallocations needed to serve a request in the worst case. In this setting, our result is a corresponding amortized analysis.

Classification: Algorithms and data structures

1 Introduction and motivation

Universal Mobile Telecommunications System (UMTS) is one of the third-generation (3G) mobile phone technologies that uses W-CDMA as the underlying standard, and is standardized by the 3GPP [7]. The W-CDMA (Wideband Code Division Multiple Access) is a wideband spread-spectrum 3G mobile telecommunication air interface that utilizes code division multiple access. The main idea behind the W-CDMA is to use physical properties of interference: if two transmitted signals at a point are in phase, they will "add up" to give twice the amplitude of each signal, but if they are out of phase, they will "subtract" and give a signal that is the difference of the amplitudes. Hence, the signal received by a particular station is the sum (component-wise) of the respective transmitted vectors of all senders in the area. In the W-CDMA, every sender s is given a *chip code* $\boldsymbol{v}$. Let us represent the data to be sent by a vector of ± 1. When s wants to send a *data vector* $\boldsymbol{d} = (d_1, \ldots, d_n)$, $d_i \in \{1, -1\}$, it sends instead a sequence $d_1 \cdot \boldsymbol{v}, d_2 \cdot \boldsymbol{v}, \ldots, d_n \cdot \boldsymbol{v}$, i.e. n-times the chip code modified by the data. For example, consider a sender with a chip code $(1, -1)$ that wants to send data $(1, -1, 1)$; then the actually transmitted signal is $(1, -1, -1, 1, 1, -1)$. The signal received by a station is then a sum of all transmitted signals. Clearly, if the chip codes are orthogonal, it is possible to uniquely decode all the signals.

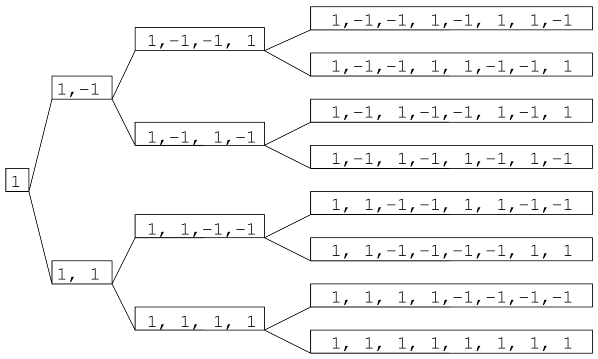


Figure 1: An OVSF tree

One commonly used method of implementing the chip code allocation is Orthogonal Variable Spreading Factor Codes (OVSF). Consider a complete binary tree, where the root is labeled by (1) , the left son of a vertex with label α is labeled (α, α) and the right son is labeled $(\alpha, -\alpha)$ (see Figure 1).

If a sender station enters the system, it is given a chip code from the tree in such a way that there is at most one assigned code from each root-to-leaf path. It can be shown [1, 6] that this construction fulfills the orthogonality property even with codes of different lengths.

Clearly, a code at depth l in the tree has length 2^l, and a sender using this code will use a fraction of $1/2^l$ of the overall bandwidth. When users enter the system, they request a code of a given length. It is irrelevant which particular code is assigned to which user, the length is the only thing that matters. When users connect to and disconnect from a given base station, i.e. request and release codes, the tree can become fragmented. It may happen that no code at the requested level is available, even though there is enough bandwidth (see Figure 2).

This problem can be solved by changing the chip codes of some already registered users, i.e. reallocating the vertices of the tree. Since the cost of a reallocation dominates this operation, the number of reallocations should be kept minimal. In [4] the authors considered the problem of minimizing the number of reallocations over a given schedule and showed that it is NP-hard to generate an optimal allocation schedule. In this paper, we show that the online version proposed in [4] can be solved in amortized complexity $O(1)$ reallocations per request. The technical parts can be found in the Appendix.

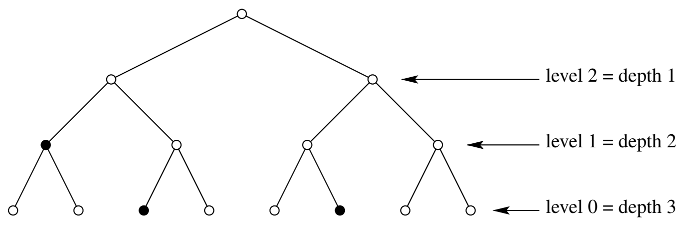


Figure 2: No code at depth 2 can be allocated although there is enough free bandwidth. Full circles represent allocated vectors.

2 Problem definition

Consider a complete binary tree $T = (V, E)$ of height n. Leaves are said to be at level 0, and the root at level n. A *request vector* is a vector $\boldsymbol{r} = (r_0, \ldots, r_n) \in \mathbb{N}^{n+1}$, where r_i represents the number of users that request a code at level i. A *code assignment* of a particular request vector $\boldsymbol{r}$ is a subset of vertices $F \subset V$, such that every path from a leaf to the root contains at most one vertex from F, and there are exactly r_i vertices at level i in F. The input consists of a sequence of requests of two types: *insertion* and *deletion*. Suppose that, after t requests the request vector is $\boldsymbol{r}_t$ and the corresponding code assignment is F_t. The algorithm has to process the next request in the following way:

A) **insertion request** at a vertex at level i.
The algorithm must output a new code assignment F_{t+1} satisfying the request vector $\boldsymbol{r}_{t+1} = (r_0, \ldots, r_i + 1, \ldots, r_n)$[1]

B') **deletion request** of a particular vertex $v \in F_t$.
Let v be at level i. The new request vector is $\boldsymbol{r}_{t+1} = (r_0, \ldots, r_i - 1, \ldots, r_n)$, and the new code assignment is $F_{t+1} = F_t \setminus \{v\}$.

During each step, the number of reassignments is $|F_{t+1} \setminus F_t|$. For a given sequence of requests $R_1, \ldots, R_m$, we are interested in the amortized number of reassignments per request, i.e. the quantity $\frac{1}{m} \sum_{t=0}^{m-1} |F_{t+1} \setminus F_t|$

First, let us note that there is no harm in allowing reallocations also in the deletion requests: in a deletion request the algorithm remembers the moves it would perform, and performs them in the next insertion request[2]. As a next step, when deleting a particular vertex v at level i, the algorithm may delete any other vertex at the same level and then reallocate the code of the deleted vertex to v. These arguments allow us to reformulate the requirements for processing deletion requests as follows:

[1] We may consider, without loss of generality, that there is enough bandwidth to satisfy each request.

[2] In case of consecutive deletion requests the algorithm can clearly maintain a mapping between the actual vertices in F and their "virtual" positions.

B) deletion request of a vertex at level i.
The algorithm must output a new code assignment F_{t+1} satisfying the request vector $\boldsymbol{r}_{t+1} = (r_0, \ldots, r_i - 1, \ldots, r_n)$.

This definition bears resemblance to memory allocation problems studied in the operating systems community, in particular to the *binary buddy system* memory allocation strategy introduced in [5]. In this strategy, requests to allocate and deallocate memory blocks of sizes 2^l are served. The system maintains a list of free blocks of sizes of 2^k. When allocating a block of size 2^l in a situation where no block of this size is free, some bigger block is recursively split into two halves called *buddies*. When a block whose buddy is free is deallocated, both buddies are recursively recombined into a bigger block.

The properties of binary buddy system have been extensively studied in the literature (see e.g. [2] and references therein). However, the bulk of this research is focused on cases without reallocation of memory blocks. E.g. [2] shows how to implement the buddy system in amortized constant time per allocation/deallocation request, but in a model where reallocation is not allowed.

A binary buddy system with memory block reallocations has been studied in [3]. This paper analyzes both the number of reallocated blocks and the number of reallocated bytes per request; the analysis of the number of reallocated block is in fact the very same model that is used in our paper. However, only the worst case scenario is dealt with in [3], hence our results are relevant also to the recent memory allocation research.

3 The algorithm

We propose an online algorithm that processes the sequence of requests according to the rules A) and B). If we order the leaves from left to right, and label them with numbers $0, \ldots, 2^n - 1$, there is an interval of the form $I_u = \langle i2^l, (i+1)2^l - 1 \rangle$ assigned to each tree vertex u of level l. We shall call each such interval a *place* of level l, and we say that I_u begins at position $i2^l$. For a given code assignment F, we shall call a place I_u corresponding to a vertex u an *empty* (or *free*) *place* if neither u nor any vertex from the subtree rooted at u is in F. If $u \in F$ the corresponding place I_u is an *occupied place*, and we say that there is a *pebble* of level l located on I_u (or, alternatively there is a pebble of level l at position $i2^l$). Since in a code assignment F, every path from a leaf to the root contains at most one vertex, we can view the code assignment F as a sequence of disjoint places which are either empty or occupied by pebble (see Figure 3). While the free places in this decomposition are not uniquely defined, we shall overlook this ambiguity, as we will argue either about a particular place or about the overall size of free places (called also *free bandwidth*).

We shall say that a pebble (or place) of level l has *size* 2^l. The left (right) neighbor of a pebble is a pebble placed on the next occupied place to the left (right)[3]. We shall denote pebbles by capital letters $A, B, \ldots, X$, and their

[3]Sometimes we talk about a left (right) neighbor of a free place.

corresponding sizes by $a, b, \ldots, x$. Sometimes we shall use the notion of a vertex and the corresponding place interchangeably.

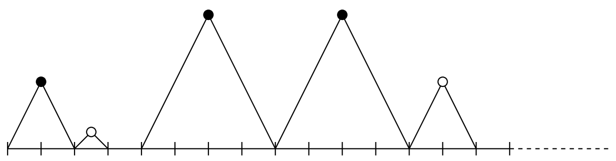


Figure 3: A code assignment seen as a sequence of black and white pebbles.

The key idea of our algorithm is to maintain a well defined structure in the sequence of pebbles. An obvious approach would be to keep the sequence sorted – i.e. the pebbles of lower levels always preceding the pebbles of higher levels with only the smallest necessary free places between them. It is easy to see that the addition and deletion to such structured sequence can be done with at most $O(n)$ reallocations.[4] Unfortunately, it is also not difficult to see that there is a sequence of requests such that this approach needs amortized $O(n)$ reallocations per request (see [4]). The problem is that a too strictly defined structure needs too much "housekeeping" operations.

The structure maintained by our algorithm will be less rigid. Basically, we will maintain a sorted sequence as in the above example, however, not all pebbles must be included in this sequence. Pebbles that form a sorted sequence according to the previous rule will be called *black* pebbles. There can be also *white* pebbles present that do not fit into this structure but we impose other restrictions on them. Informally, if there is a free place in the black sequence, a single white pebble can be placed at the beginning of this place. Moreover, all white pebbles form an increasing sequence. Formally, the structure of the sequence of pebbles is described by the following invariants. It can be shown that they indeed imply the informal description above.

C: Pebble A is *black*, if there is no bigger pebble before A, otherwise it is *white*

P1: The free bandwidth before any pebble X is strictly less than x.

P2: There is always at least one black pebble between any two white pebbles.

P3: There is no white pebble such that both its left and right neighbor are black and of the same size.

A sequence of pebbles and free places satisfying **P1**–**P3** will be called a *valid situation*. The key observation about valid situations is that in a valid situation it is always possible to process an insertion request without reallocations:

[4]The addition request is processed by placing the pebble of level l at the end of sequence of pebbles of this level. If this place is already occupied by some pebble B, B is removed and reinserted. Since there are at most n different levels, and the levels of reinserted pebbles are increasing, the whole process ends after $O(n)$ iterations. The deletion works in a similar fashion.

Lemma 1 *Consider a valid situation, and an insertion request of size $a = 2^l$. If there is a free bandwidth of at least a, then there is also a free place of size a.*

Our algorithm maintains valid situations over the whole computation, so all requests can be processed without reallocations. However, processing a request may result in a situation that is not valid. The most difficult task is to develop a post-processing phase in each request that restores the validity using only few reallocations. We show that in the case of insertion requests, a constant number of reallocations in each request is sufficient. In the deletion requests, however, a more involved accounting argument is used to show that the *average* number of reallocations per request in a worst-case execution remains constant.

3.1 Procedure Insert

Before presenting the procedure for managing insertion requests, we need an additional definition.

Definition 1 *The* closing position *of level l (of size x) is the position after the last black pebble of level l (of size x) if such pebble exists. Otherwise, the closing position of level l is the position of the first pebble of level bigger than l (size bigger than x).*

Algorithm 1 Procedure Insert: inserts a pebble A of size a into a valid situation.

```
let $P$ be the first free place of size $a$
let $B$ be the left neighbor of $P$
if $B$ does not exist or $B$ is black then
    put $A$ on $P$
    return
end if

let $C$ be the left neighbor of $B$
if $c < a$ then
    put $A$ on $P$
    return
else if $c = a$ then
    remove $B$
    put $A$ just after $C$
    put $B$ just after $A$
    return
end if

remove $B$
if $a < b$ then
    rename $A$ and $B$ so that $A$
     is the bigger pebble
end if

let $D$ be the pebble at the closing
position of size $a$.
if $D$ is white then
    let $E$ be $D$'s right neighbor
    remove $D$, $E$
    put $A$ at $D$'s original position
    put $B$ after $A$
    put $D$ after $B$
    put $E$ at $B$'s original position
else if $D$ is black then
    remove $D$
    put $A$ at $D$'s original position
    put $B$ after $A$, put $D$ after $C$
end if
```

Procedure Insert processes a new request of size a in a valid situation. First, a free place of size a is found, and a pebble is put on this place. If the sequence is no longer valid after this operation, the algorithm reassigns a constant number of pebbles in order to restore the invariants. The procedure is listed as Algorithm 1, and its analysis is given in the following theorem:

Theorem 1 *Consider a sequence of pebbles and free places that forms a valid situation, and an insertion request of size a. If there is enough bandwidth, procedure* Insert *correctly processes the request. Moreover, after finishing, the situation remains valid, and only a constant number of pebbles has been reassigned.*

Sketch of proof: Here, we present an overall structure of the proof with a number of unproven claims. The complete version can be found in Appendix.

Consider an insertion request of size a, and suppose the invariants hold. Because of Lemma 1 the line 1 of Algorithm 1 is correct, i.e. there is a free place of size a. Let P be the first such free place. If P does not have a left neighbor (i.e. there is no other pebble present), the algorithm puts A on P in line 4 and exits. From now on suppose that P has a left neighbor B, and denote the potential right neighbor of P by Z. The rest of the proof consists of a case analysis of a number of cases as they follow from Algorithm 1. For each case, the action of the algorithm is analyzed and it is proven that the resulting situation is valid.

The easy part is when B is black, since in this case the algorithm puts A on P in line 4 and exits. If B is white, however, there exists a left neighbor C of B, such that C is black and $c > b$. We distinguish three sub-cases: $c < a$, $c = a$, and $c > a$. The case $c < a$ is handled by putting A on P in line 10. If $c = a$, the sequence of lines 13–16 is executed: first, pebble B is temporarily removed. Since there has been no other pebble between A and C, and $a = c$, the place of size a immediately following C is now free. Put A immediately after C. Since $b < c = a$, the place of size b immediately following A is now free, so B can be put there. Finally, if $c > a$, the algorithm temporarily removes B[5], and calls the pebble at the closing position of size a by D (there must be a pebble present). The proof is concluded be considering two final sub-cases based on whether D is white or black. In the first case, the action is depicted on Figure 4.

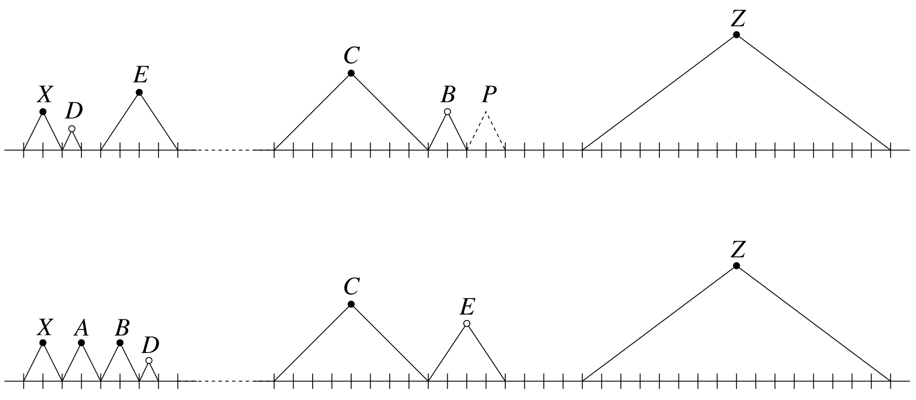


Figure 4: An example of executing lines 26–31 of procedure Insert

If D is black, the situation is as follows from Figure 5.

□

[5]assume w.l.o.g. that $a \geq b$, see Appendix

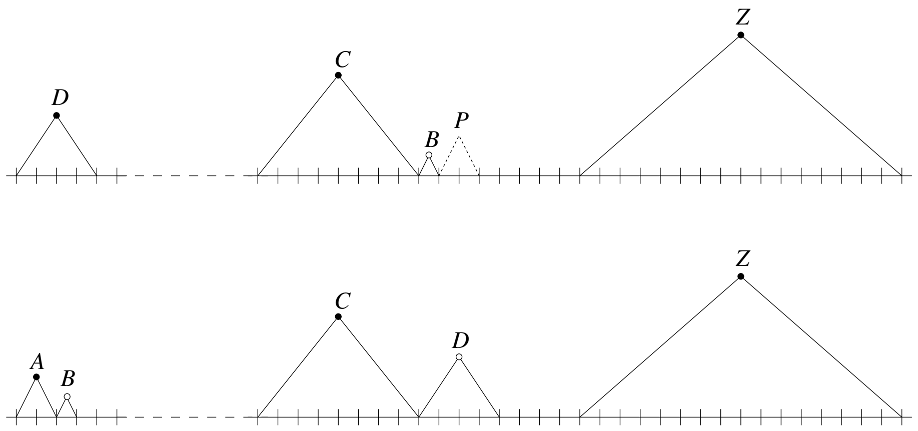


Figure 5: An example of executing lines 33–35 of procedure Insert

3.2 Procedure Delete

Algorithm 2 Procedure Delete that removes the last pebble of level l.

```
1: let $A$ be the last pebble of level $l$, and $i$ be the starting position of $A$
2: remove $A$
3: while there are any pebbles to the right of $i$ do
4:     let $x$ be the size of the smallest pebble to the right of $i$
5:     if there is a free place of size $x$ starting at $i$ then
6:         let $X$ be the rightmost pebble of size $x$
7:         let $j$ be the starting position of $X$
8:         move $X$ to $i$
9:         if $X$ has white left neighbor $Q$, and $Q$ has a left neighbor $W$ of size $x$ then
10:            swap $X$ and $Q$
11:        end if
12:        let $i$:=$j$
13:    else
14:        exit
15:    end if
16: end while
```

The deletion request requires to remove one pebble of a specified level. Procedure Delete (see Algorithm 2) first removes the last (rightmost) pebble A of the requested level. However, it may happen that this action violates the invariants **P1**–**P3**. To remedy this, the algorithm uses several iterations to "push the problem" to the right. The "problem" in this case is in the free place caused by removing A. The "pushing" is done by selecting a suitable pebble X to the right of A, removing it, and using it to fill in the gap. A new iteration then starts to fix the problem at X's original place. The suitable candidate is found as follows: from among all pebbles to the right of A, select the smallest one. If there are more pebbles of the smallest size, select the rightmost one. To argue the correctness it is needed to show that this procedure is well defined, and that after a finite number of iterations, a valid situation is obtained (the full proof can be found in Appendix):

Theorem 2 *Consider a sequence of pebbles and free places that forms a valid situation, and a deletion request of size a. Procedure* Delete *correctly processes the request. Moreover, after finishing, the situation remains valid.*

Sketch of proof: Let us number the iterations of the **while** loop by $t = 1, 2, \ldots$. Let Γ_t be the configuration of pebbles and free places at the beginning of the t-th iteration. Hence, the t-th iteration starts with Γ_t, and a position i_t; it selects a pebble X_t of size x_t starting at j_t, moves it to i_t, and sets $i_{t+1} := j_t$. Moreover, at the beginning of the t-th iteration we shall consider a free place P_t of size a_t starting at i_t, such that $a_1 = a$, and the size a_{t+1} is determined as follows: let Γ'_t be obtained from Γ_t by putting a new pebble A_t on P_t. If the color of X_t in Γ'_t is black then let $a_{t+1} = x_t$, otherwise let Y_t be X_t's left neighbor, and $a_{t+1} = y_t$. It can be shown that this definition is correct, i.e. that P_{t+1} is indeed free.

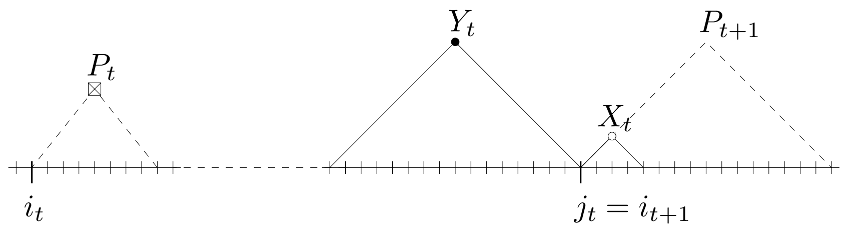


Figure 6: One iteration of the **while** loop in procedure Delete.

We shall prove the following claim by induction on t:

For every iteration t of the **while** *loop, the corresponding Γ'_t is a valid situation.*

The first iteration starts after removing A from a valid situation, so the claim for $t = 1$ holds.

Consider the t-th iteration. The algorithm either stops or enters the next iteration. We prove that in the latter case Γ'_{t+1} is valid, provided Γ'_t was valid. The proof again continues with a case-analysis. First, if $x_t < a_t$, it is possible to prove that there are neither white pebbles nor free places between i_t and j_t, from which the invariants readily follow. On the other hand, if $x_t \geq a_t$, then the swap on line 10 never happens, and it is again possible to argue the validity of the resulting situation.

Having proved the claim, the proof is concluded by analyzing the last iteration: the algorithm can stop either when there are no pebbles to the right of i_t, or when the selected pebble X_t does not fit to position i_t. In both cases it is possible to show that the resulting situation is valid.

□

4 Complexity

So far we have argued about the correctness of the algorithm, showing that it correctly processes all requests. This section is devoted to the analysis of the average number of reassignments per request needed in the worst-case computation. Obviously, the only situation in which a non-constant number of reallocation could be performed is the iteration of the **while** loop in DELETE. Hence, our aim is to develop an accounting scheme that would ensure a linear (in the number of requests) number of iterations of the **while** loop over the whole computation. To this end we introduce the notion of *coins*: each request has associated a constant number of coins which can be put on some places. Every iteration of the main loop in DELETE consumes a coin. In the following we show where to put the constant number of coins in every request such that each iteration of the loop in DELETE can be paid by an existing coin.

In our coin placement strategy we shall maintain the following additional invariant:

P4: Consider a free place P such that there are some pebbles to the right of P. Let X be the smallest pebble to the right of P. Then there are at least $\lfloor 2p/x \rfloor$ coins on P.

From now on we shall consider situations with some coins placed in some places, and we show how to manage the coins so that there is always enough cash to pay for each iteration in DELETE. The following two lemmas present the accounting strategy for INSERT and DELETE:

Lemma 2 *Let us suppose that procedure* INSERT *was called from a situation in which invariants* **P1**– **P4** *hold. Then it is possible to add a constant number of coins and reallocate the existing ones in such a way that invariants* **P1**– **P4** *remain valid.*

Proof: It has already been proven that invariants **P1**– **P3** are preserved by procedure INSERT, so it is sufficient to show how to add a constant number of coins in order to satisfy **P4**. During procedure INSERT, only a constant number of pebbles are *touched* – i.e. added or reassigned. Let Γ be the situation before INSERT and Γ' be the situation after INSERT finished. Let P be a free place in Γ' and X be the smallest pebble to the right of P. We distinguish two cases:

Case 1: X was touched during INSERT

If $p < x/2$, no pebbles are required on P, so let us suppose that $p \geq x/2$. However, since Γ' is valid, the free bandwidth before X is less than x, so $p = x/2$, and there is only one pebble required in P; this pebble will be placed on P and charged to X. Obviously, for each touched pebble X, there may be only one free place of size $x/2$ to the left (because of **P1**), so every touched pebble will be charged at most one coin using in total constant number of coins.

Case 2: X was not touched during INSERT

If P was free in Γ the required amount of $\lfloor 2p/x \rfloor$ coins was already present on P in Γ, so let us suppose that P was not free in Γ. That means that P became free in the course of INSERT when some pebbles were reallocated. Using similar arguments as in the previous case we argue that $p = x/2$. However, during INSERT only a constant number of free places of a given size could be created, so it is affordable to put one coin on each of them.

□

Lemma 3 *Let us suppose that procedure* Delete *was called from a situation in which invariants* **P1**– **P4** *hold. Then it is possible to add a constant number of coins, remove one coin per iteration of the main loop, and reallocate the remaining coins in such a way that invariants* **P1**– **P4** *remain valid.*

Sketch of proof: Let us suppose that there is at least one full iteration of the main loop. Recall the notation from the proof of Theorem 2, i.e. we number the iterations of the main loop, and Γ_t is the configuration at the beginning of t-th iteration. Moreover, Γ'_t is obtained from Γ_t by putting a new pebble A_t on P_t. From the proof of the theorem it follows that Γ'_t is always a valid situation. We prove by induction on t that the following can be maintained:

For every iteration $t > 1$ of the **while** *loop, the corresponding Γ'_t satisfies* **P1**–**P4**, *all pebbles to the right of i_t have size at least a_t, and one extra coin lays on A_t. Moreover, if some pebble to the right of i_t has the size a_t, then two extra coins lay on A_t.*

We omit the details about the induction basis, and proceed with the induction step. In order to prove the claim for Γ'_{t+1}, consider the situation when the algorithm finishes the t-th iteration and enters the $t+1$st. Γ'_{t+1} is obtained from Γ'_t by removing A_t, moving X_t to i_t, and placing A_{t+1} on $i_{t+1} = j_t$. Note that in this case $x_t \geq a_t$, so there is no swap. It is possible to show that all pebbles to the right of i_{t+1} have size at least a_{t+1}, and that there is no free place between i_t and j_t in Γ'_{t+1}. Since for the free places before i_t and after j_t, **P4** remains valid, we argue that **P4** holds in Γ'_{t+1}.

Now we show how to find two free coins – one to pay for the current iteration, and one to be placed on A_{t+1}. If $x_t = a_t$, there are two coins placed on A_t. Otherwise (i.e. in case that $x_t > a_t$) one coin comes from the deletion of A_t and the other can be found as follows. Since X_t was placed on i_t, and $x_t > a_t$, there must have been a free place P of size $x_t/2$ in Γ'_t. Moreover, X_t was to the right of P, and so there must have been at least one coin on P. In Γ'_{t+1}, P is covered by X_t, so the coin can be used.

The last thing to show is to find a second free coin in case that there exists some pebble Q to the right of A_{t+1} of size $q = a_{t+1}$. In this case X_t is white in Γ'_t – otherwise it would hold that $a_{t+1} = x_t$ and there would be no pebble of size x_t to the right of X_t. Hence $x_t \leq a_{t+1}/2$ and A_{t+1} in Γ'_{t+1} covers a place of size $a_{t+1}/2$ that has been free in Γ'_t. According to **P4** there is a coin on this place in Γ'_t; this coin can be used.

The proof of the theorem is concluded by considering the last iteration. Let $\Gamma'_{t_{fin}}$ be the last situation. The final situation is obtained from $\Gamma'_{t_{fin}}$ by removing $A_{t_{fin}}$. We show that **P4** holds. The only free place that could violate **P4** is the one remained after $A_{t_{fin}}$, however, there was a coin on $A_{t_{fin}}$, and all pebbles to the right (if any) are bigger than $a_{t_{fin}}$.

□

5 Conclusion

We have presented an online algorithm for bandwidth allocation in wireless networks, which can be used to perform the OVSF code reallocation with the amortized complexity of $O(1)$ reallocations per request. This is an improvement over the previous best known result achieving the competitive ratio of $O(n)$. Moreover, the constant in our algorithm is small enough to be of practical relevance.

On the other hand, no attempt has been made at minimizing this constant. With the best known lower bound of 1.5 it would be worthwhile to close the gap even further.

References

[1] F. Adachi, M. Sawahashi, and K. Okawa. Tree structured generation of orthogonal spreading codes with different lengths for the forward link of DS-CDMA mobile radio. *IEE Electronic Letters*, 33(1):27–28, 1997.

[2] G. S. Brodal, E. D. Demaine, and J. I. Munro. Fast allocation and deallocation with an improved buddy system. *Acta Informatica*, 41(4–5):273–291, March 2005.

[3] D. C. Defoe, S. R. Cholleti, and R. K. Cytron. Upper bound for defragmenting buddy heaps. In *LCTES '05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 222–229, New York, NY, USA, 2005. ACM Press.

[4] T. Erlebach, R. Jacob, M. Mihalák, M. Nunkesser, G. Szabó, and P. Widmayer. An algorithmic view on OVSF code assignment. In V. Diekert and M. Habib, editors, *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 2004.

[5] K. C. Knowlton. A fast storage allocator. *Communications of ACM*, 8(10):623–624, 1965.

[6] T. Minn and K.-Y. Siu. Dynamic assignment of orthogonal variable-spreading-factor codes in W-CDMA. *IEEE Journal on Selected Areas in Communications*, 18(8):1429–1440, 2000.

[7] Wikipedia. Universal mobile telecommunications system. Available at: `http://en.wikipedia.org/wiki/Umts`. From Wikipedia, the free encyclopedia [Online; accessed 23. March 2006].

Appendix

This part contains the technical parts and proofs excluded from the previous sections. Let us first present a few structural lemmata describing valid situations. Informally, we prove that a valid situation (i.e. situation satisfying invariants **P1**–**P3**) can be described as follows: assign black color to pebbles that are not preceded by a bigger pebble. Then these pebbles form a non-decreasing sequence with as few free places between pebbles as possible. In each free place of this sequence, at most one white pebble may be present, aligned to the left. Moreover, white pebbles form a strictly increasing sequence.

Unless stated otherwise, the following lemmas assume a valid situation.

Observation 1 *The sizes of black pebbles form a non-decreasing subsequence.*

Observation 2 *If the last pebble is removed, the invariants* **P1**–**P3** *remain true, i.e. the situation remains valid.*

Lemma 4 *In a non-empty valid situation, the first pebble is placed at position 0.*

Proof: Consider, for the sake of contradiction, the first pebble A of size a. Clearly, A must be at position ia for some $i > 0$. But then there is some free place of size a before A – contradiction with **P1**.

□

Lemma 5 *All black pebbles of a given level l occupy consequent places.*

Proof: Let A and B be two black pebbles of level l such that $a = b = 2^l$, A being on the left. Let X be a pebble between A and B. Because B is black, $x \leq b$ holds. From **P3** it follows that X must be black, i.e. $x \geq a$. Hence, X is black and of the same size as A and B. That means that the black pebbles of level l form a sequence that can be interrupted by some free places but not by pebbles of different sizes.

Consider now, for the sake of contradiction two black pebbles A and B of level l separated by some free places. Because the positions and sizes of A and B are multiples of 2^l, it follows that if they are separated by some free places, there is also a free place of level l between them. However, this would contradict **P1**.

□

Lemma 6 *The left neighbor of a white pebble A always exists, is black, and strictly bigger than A. Moreover, there is no free space between A and its left neighbor.*

Proof: Consider a white pebble A. Since there is a bigger pebble before A, there must be a left neighbor B. Because of **P2**, B is black. If $b \leq a$ then A would be black, too, so B must be bigger than A.

Let A and B be separated by some free places. Since b is a multiple of a, there must be a free place of size a between them – a contradiction with **P1**.

□

Corollary 1 *If there is a free place before a pebble A, then A is black.*

Lemma 7 *Consider a pebble A followed by a free place. Then the overall size of these free places is $p \geq a$.*

Proof: Let A be a pebble of level l followed by some free place P. If A is the last pebble, then A is followed by a place of level l which is free.

Let B be the right neighbor of A. Because of Corollary 1, B must be black, i.e. $b \geq a$. Since the positions and sizes of both A and B are multiples of $a = 2^l$, the free place following A has to be at least of size a.

□

Lemma 8 *Consider a black pebble B followed by a white pebble A. Then A is followed by free places of overall size at least $b - a \geq b/2 \geq a$.*

Proof: Because of Lemma 6, there is no free space between A and B, and $b \geq 2a$. If A is the last pebble, the proposition follows immediately.

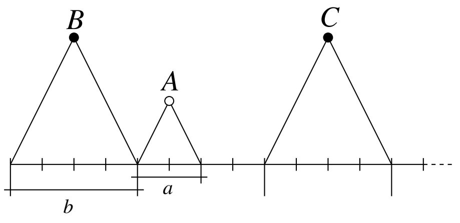


Figure 7: Situation in the proof of Lemma 8.

If A is not the last pebble then there exists its right neighbor C (this situation is depicted in Figure 7). Because of **P2**, C is black, i.e. $c \geq b$. As the positions and sizes of B and C are multiples of b, the distance between them is at least b. Because A immediately follows B, and there are no other pebbles between B and C, the overall size of free spaces following A is at least $b - a$, and the proposition follows.

□

Lemma 9 *The sizes of white pebbles form a strictly increasing subsequence.*

Proof: Consider a white pebble A, and a white pebble B located somewhere after A. Consider, for the sake of contradiction, that $a \geq b$. Because of Lemma 8, the place P of size a immediately following A is free. Obviously, B is located after P which is a contradiction with **P1**.

□

The following two lemmas follow directly from the structure of the valid situations, and provide a useful tool in proving the correctness of the algorithm:

Lemma 10 *Consider a black pebble B immediately followed by a white pebble A. Then there is no white pebble $X \neq A$ such that $a \leq x < b$.*

Proof: Consider, for the sake of contradiction, a white pebble X satisfying $a \leq x < b$. Because of Lemma 9, X cannot be located before A. However, according to Lemma 8, there are free spaces of overall size at least $b/2$ immediately following A. Because $x < b$, it holds $x \leq b/2$, and **P1** is violated.

□

Lemma 11 *For a given l, let P be the first free place of size $p = 2^l$. Then the free bandwidth before P is strictly less than p.*

Proof: First consider the case when all pebbles to the left of P are of size at least p. Since all the positions and sizes of those pebbles are multiples of p, there is either no free place before P, or a free place of level at least l. However, the latter case would contradict the fact that P is the first free place of size p.

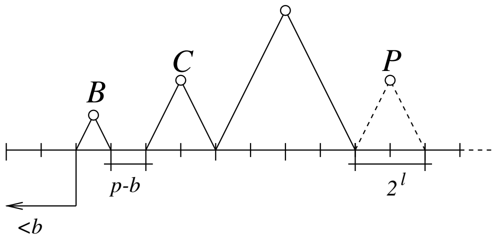


Figure 8: Situation in the proof of Lemma 11: there is a smaller pebble B to the left of P.

Let B be the first pebble to the left of P such that $b < p$, i.e. there are only pebbles of size at least p between B and P (see Figure 8). From **P1** it follows that the free bandwidth before B is strictly less than $b \leq p/2$. If there are any pebbles between B and P, let C be B's right neighbor. Using the same argument as above we can argue that there is no free place between C and P. What remains to be shown is that the overall size of free places between B and P immediately following B is at most $p - b$. Denote this size as α and let us suppose, for the sake of contradiction, that $\alpha > p - b$. Since the positions and sizes of P and all pebbles between B and P are multiples of p, it follows that $\alpha \geq p$. The reason why it is so is clear from Figure 9. Consider the places of

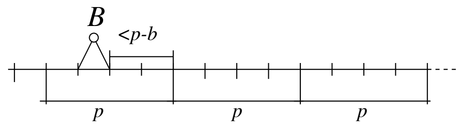


Figure 9: Situation in the proof of Lemma 11: if $\alpha > p - b$, there is a free place of size p.

size p: B must be fully located in one of them, and the remaining free places within this place have overall size at most $p - b$. Since $\alpha > p - b$ there must be some free place in the next place of size p. However, since all subsequent pebbles are of size at least p, they start at the beginning of a place of size p.

Hence the next place of size p is free – a contradiction with the fact that P is the first free place of size p.

□

With the developed machinery we are able to prove the crucial Lemma 1, stating that in valid situations, if there is enough free bandwidth to satisfy an insertion request of size a, there is always a free place of size a.

Proof of Lemma 1 Consider, for the sake of contradiction, that there is no free place of size a. Let P be the last place of size a, i.e. the interval $\langle 2^n - a, 2^n - 1\rangle$. Obviously, P is not free. If there is a pebble B of size $b < a$ inside P then because of **P1**, the free bandwidth before B is strictly less than b. However, the free bandwidth after B is at most $a - b$, hence there is strictly less than a free bandwidth overall.

So it must be that P is not free because it is a part of a pebble X of size at least a. Following Observation 2, X can be removed and the invariants still hold. Now consider the first free place A of size a (it is at the beginning of the removed pebble X): because of Lemma 11, there is strictly less than a free bandwidth before A. However, this is all free bandwidth that exists in the original situation.

□

Another observation that is useful in the case-analysis of the proof of Theorem 1 is the following lemma:

Lemma 12 *Consider a sequence of pebbles and free places that forms a valid situation. Let P be the first free place of size p. Let Z be the right neighbor of P. If Z exists, it is black and $z > p$.*

Proof: Since P is a free place and Z is P's right neighbor, there is a free place immediately preceding Z. According to Lemma 6 Z cannot be white. The fact that $z > p$ follows immediately from **P1**.

□

Now we are ready to prove the correctness of procedure INSERT:

Proof of Theorem 1 Consider an insertion request of size a, and suppose the situation is valid. Because of Lemma 1 the line 1 of Algorithm 1 is correct, i.e. there is a free place of size a. Let P be the first such free place.

If P does not have a left neighbor, the algorithm puts A on P in line 4 and exits. Since there is no pebble to the left of P and P is the first free place if size a, it means that P starts at position 0. However, due to Lemma 4 it follows that there are no other pebbles, and the situation is valid. From now on suppose that P has a left neighbor B, and denote the potential right neighbor of P by Z. Here we present the full case analysis.

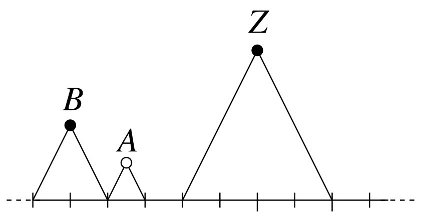
Case 1: B is black

The algorithm puts A on P in line 4 and exits. Obviously, no pebble to the left of A changes color. If there are some pebbles to the right of A, then because of Lemma 12 there exists a black Z, such that $z > a$ (see Figure 10). Hence, no pebble changes color after A is put on P. We prove that all invariants hold:

P1: For A it follows from Lemma 11. For all other pebbles the free bandwidth before them could have only been decreased.

P2: Since B is black, the only way **P2** could be violated is if Z exists and is white. However, due to Lemma 12 it is not possible.

P3: Consider, for the sake of contradiction, that after A was put on P, the invariant **P3** is violated. Assume that A is white. In this case Z exists, is black and $z = b$, which contradicts Lemma 12. On the other side if A is black, then B or Z must be white. This contradicts Lemma 12, too.



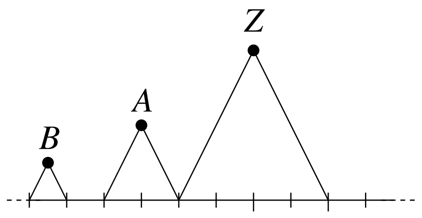


Figure 10: Two examples of Case 1

Case 2: B is white

Because of Lemma 6, there exists a left neighbor C of B, such that C is black and $c > b$. Hence, line 8 is correct. We again distinguish three cases according to the relation of c and a.

Case 2.1: $c < a$

This case is handled by putting A on P in line 10. Because C is black and $b < c < a$, A is colored black. Obviously, no pebble before A changed color. If there are some pebbles to the right of A, then because of Lemma 12 there exists a black Z, such that $z > a$ (see Figure 11). Hence, no pebble changes color after A is put on P.

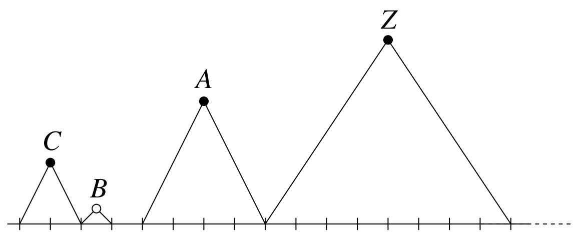


Figure 11: Case 2.1

We prove that the situation remains valid, i.e. the invariants **P1**–**P3** remain true.

P1: The same argument as in **Case 1**.

P2: Holds trivially, since only a black pebble A has been added.

P3: Consider, for the sake of contradiction, that after A was put on P, there is a white pebble W between two black pebbles of the same size. Since A is black, $A \neq W$, i.e. W is a neighbor of A. Since Z is black, we have $W = B$. As $c < a$, invariant **P3** holds – a contradiction.

Case 2.2: $c = a$

This situation is handled in the block on lines 13–16: first, pebble B is temporarily removed. Since there has been no other pebble between A and C, and $a = c$, the place of size a immediately following C is now free. Put A immediately after C. Since $b < c = a$, the place of size b immediately following A is now free, so B can be put there (see Figure 12). Since C is black, A is black, too. Using arguments as in Case 2.1 we argue that no pebble has changed its color.

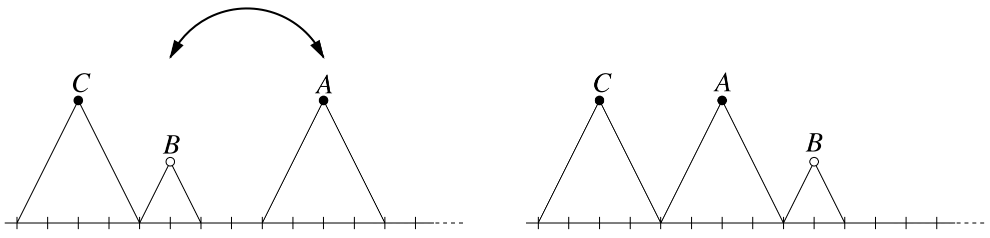


Figure 12: Case 2.2

Again, we prove that after the reassignment all invariants hold.

P1: Suppose that A was put on P. The invariant holds using the same argument as in Case 1. The reassignment changed the status of pebbles A and B only. However, the free bandwidth before B has not changed, and the free bandwidth before A could have only been decreased.

P2: Suppose for the sake of contradiction that after the reassignment of pebbles there are two neighboring white pebbles. Clearly, one of them must be B. However, A is black so it must be that B's right neighbor is white. The contradiction follows from Lemma 12 – B's right neighbor is the original P's right neighbor Z, which is black.

P3: Suppose that after the algorithm finishes, there is a white pebble $W \neq A$ between two black pebbles of the same size. The only candidate for W is the pebble B. Due to Lemma 12, if the right neighbor of B exists, $z > a$ holds, which is a contradiction.

Case 2.3: $c > a$

The algorithm starts this case by temporarily removing B. First, we argue that we can without loss of generality consider $a \geq b$. Suppose that $a < b$, i.e. the situation is as on Figure 13 left. We will treat this situation exactly as if a and b would be swapped, i.e. as if the requested size was b and the existing pebble B' was of size a (Figure 13 right) – the removal of B and B' will render the same situation.

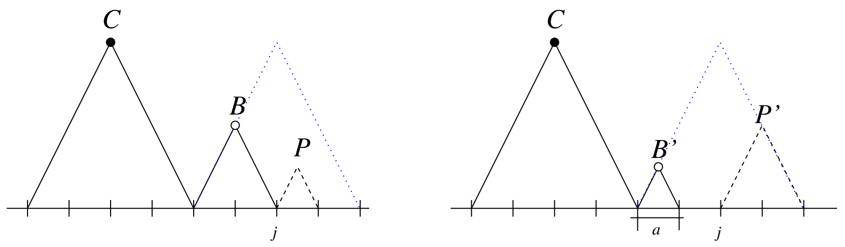


Figure 13: Case 2.3: the situation $a < b$ can be transformed to $a \geq b$.

However, we have to show that if we replace the pebble B of size b by a pebble B' of size a, the situation remains valid (i.e. **P1**–**P3** still hold). Clearly, the colors of all pebbles remain the same, and hence **P2**, **P3** remain true. **P1** remains true because Lemma 11 states that the free bandwidth before P (and hence before C) is less than a. Let j be the position where P starts. What remains to be shown is the fact that the first free place of size b starts at j. Since $c > b > a$, and there is no free space between C and B and between B and P, both B and P fit into a place of size c immediately following C. However, due to Lemma 8, the mentioned space of size c does not contain other pebbles than B. Since $b \leq c/2$, there is a free place P' of size b starting at j. P' is the first free place of size b: because of Lemma 11, the free bandwidth before P (and hence before C) is less than a; the free bandwidth between B' and P' is $b - a$, hence no free space of size b exists before P'.

From now on let us suppose $a \geq b$. Let i be the closing position of size a. We argue that there must be a pebble at i. To see why recall the definition of closing position: either there is a pebble of size a ending at $i - 1$, or a pebble of size $> a$ starting at i. Hence, if there is no pebble at position i, there is a pebble of size a just before i, and due to Lemma 7 there is a free space of size a immediately following it. However, i is to the left of C – a contradiction with the fact that P was chosen to be the first free place of size a. Let us denote the pebble at position i by D, and once more distinguish two sub-cases:

Case 2.3.1: D is white

From the definition of the closing position it follows that D is a white pebble immediately preceded by a black pebble X of size a. Obviously, D has a right neighbor E which is, due to **P2** black[6], and due to **P3** it holds $e > a$. Moreover, it holds that $a = b$: we argued above that $a \geq b$. Because of Lemma 10 applied to X and D, and Lemma 9, it holds that $b \geq a$.

The action of the algorithm in this case is depicted on Figure 4: E is moved after C, and A, B are inserted between X and D. We first prove that this operation is correct, i.e. there is always an appropriate free place to put the pebbles.

After D and E were removed, the three places of size a immediately following X are free; the reason is that $e \geq 2a$, i.e. E's starting position was a multiple of a: the first of the three places originally contained only D, and the remaining

[6]as a special case, it might be $E = C$

two were occupied by E. Hence, A, B, and D can be placed after X. Moreover, after B was removed, the free place of size e immediately following C is free – if Z exists, it is due to Lemma 12 black, and so its starting position is a multiple of c. Since $e \leq c$, it follows that E can be placed at B's original position. We now make sure that the situation is valid, i.e. the invariants **P1**–**P3** hold. First, let us argue about the colors of the pebbles: pebbles to the left of X (including X) don't change color, and neither do pebbles to the right of Z (including Z). Because $a = b$, pebbles A and B become black. Pebble D remains white. Since $c > a$, C remains black. E may be either white, if $e < c$, or black, if $e = c$.

Lemma 10 applied to B and C ensures that there are no white pebbles of size between a and c. Since $a < e \leq c$, all pebbles between E's original position and C were black. Hence, they remain black also after the reassignment. Now, let us argue about the invariants:

P1: For pebble X and all pebbles to the left **P1** holds trivially. For Z and pebbles to the right the free bandwidth before them decreased when A was added. For A and B, **P1** holds because Lemma 11 asserts that the free bandwidth before P is less than a, and they were placed to the left of P. For D, the free bandwidth to the left did not change.

Consider the new position of pebble E. Since E was placed at B's original position, the bandwidth before E is the original free bandwidth before B (due to Lemma 11 at most a) increased by e (E was removed), and decreased by $2a$ (A and B were inserted). Hence, we get that the free bandwidth before E is at most $a + e - 2a < e$, and **P1** holds for E.

As we argued before, all pebbles between E's original position and E's new position are black, and bigger or equal than E, thus **P1** holds for them, too.

P2: Violating **P2** means that there are two consecutive white pebbles. Obviously, if such two pebbles exist they must be between X and Z because the original position was valid. However from these pebbles, only D and E may be white, and only if $e < c$. But in this case there is a black pebble $C \neq E$ between E and D.

P3: Again, only D and E are candidates for violating **P3**. If Z exists, $z > c$, because originally there was white B between them. Hence E cannot violate **P3**. Originally, X was the last black pebble of size a, so D cannot violate **P3**, either.

Case 2.3.2: D is black

In this case the definition of closing position ensures that D is the first black pebble of size bigger than a, and that there are no black pebbles of size a. Thus we get the following inequalities

$$b \leq a < d \leq c$$

As a consequence of Lemma 10 applied to B and C we get that all pebbles between D and C are black (with a possible special case $D = C$).

The action of the algorithm is described on Figure 5: pebble D is removed, and pebbles A and B are placed into the free space. Pebble D is then put at B's original position. First, we reason that this action is well defined. Since both $a, b \leq d/2$, pebbles A and B fit into the free space created by removing D. Moreover, the place of size c immediately following C is, after removing B, free: if Z exists, it starts at a position that is a multiple of c. Since $d \leq c$, D fits into this position.

Now consider the colors of the pebbles: pebble A becomes black, since D was the first (black) pebble of size bigger than a. B becomes either black or white, and so does D. The colors of all other pebbles do not change. Again, let us argue about the invariants:

P1: For A, **P1** holds because, due to Lemma 11 the free bandwidth before P was at most a. For pebbles to the left of A nothing changed. For Z and all pebbles to the right the free bandwidth decreased, and so it did for B.

Consider the free bandwidth before D: because of **P1**, the free bandwidth before the original position of B was less than b. Hence the free bandwidth before D is less than $b + d - a - b < d$. All pebbles between D's original position and D's new position are black and bigger than D so **P1** holds for them, too.

P2: The only possibility to violate **P2** is that B and D are consecutive white pebbles. However, B becomes white only if there is a black $C \neq D$ before it, because $d > a$.

P3: Since $z > c$, **P3** cannot be violated by D. Moreover, $a < d$ so **P3** cannot be violated by B.

□

The following lemma is used in the proof of correctness of Delete:

Lemma 13 *Consider a valid situation with a black pebble B starting at location i, immediately followed by a white pebble A. Let j be the closing position of size $2a$. Then no free place and no white pebble starts at any location between j and i.*

Proof: First let us prove that there is no free place between j and i. Consider, for the sake of contradiction the leftmost free place starting between j and i. Such a free place is not unique, so let P be the one with the biggest level among them. If P starts at j, then, by the definition of closing position, it is immediately preceded by a black pebble of size $2a$. It follows from Lemma 7 that P is of size at least $2a$ – a contradiction with **P1** applied to A. Hence, P must start to the right of j, and is immediately preceded by a pebble X (because it is leftmost). If X is black, then $x \geq 2a$, and the same contradiction as above follows. If X is white, due to Lemma 9, $x < a$. Moreover, due to Lemma 6, X is immediately preceded by a black pebble Y, such that $y \geq 2a$. Finally, due to Lemma 8 there is at least $y - x > a$ free bandwidth immediately following X – a contradiction.

Finally, we argue that there is no white pebble between j and i. Due to Lemma 8, a free place immediately follows each white pebble – a contradiction.

□

The complete proof of correctness of Delete is presented next.

Proof of Theorem 2 Let us number the iterations of the **while** loop by $t = 1, 2, \ldots$. Let Γ_t be the configuration of pebbles and free places at the beginning of the t-th iteration. Hence, the t-th iteration starts with Γ_t, and a position i_t; it selects a pebble X_t of size x_t starting at j_t, moves it to i_t, and sets $i_{t+1} := j_t$. Moreover, at the beginning of the t-th iteration we shall consider a free place P_t of size a_t starting at i_t, such that $a_1 = a$, and the size a_{t+1} is determined as follows: let Γ'_t be obtained from Γ_t by putting a new pebble A_t on P_t. If the color of X_t in Γ'_t is black then let $a_{t+1} = x_t$, otherwise let Y_t be X_t's left neighbor, and $a_{t+1} = y_t$. We soon prove that this definition is correct, i.e. that P_{t+1} is indeed free.
We shall prove the by induction on t the following claim:

For every iteration t of the **while** *loop, the corresponding Γ'_t is a valid situation.*

The first iteration starts after removing A from a valid situation, so the claim for $t = 1$ holds.

Consider the t-th iteration. The algorithm either stops or enters the next iteration. We prove that in the latter case Γ'_{t+1} is valid, provided Γ'_t was valid. Γ'_{t+1} is obtained from Γ_t by moving X_t to i_t, and putting A_{t+1} to P_{t+1}. Since Γ'_t is valid, the place P_{t+1} exists, and obviously is free (either directly or due to Lemma 8).

First note that the swap operation on line 10 is correct: since X_t was to the right of i_t in a valid situation Γ'_t, the free bandwidth before i_t is at most x_t. Hence, swapping X and Q yields a situation where X follows immediately after W, and Q starts immediately after X (on i_t). Now we distinguish two cases:

Case 1: $x_t < a_t$

In this case X_t is white in Γ'_t (it has a larger pebble A_t to the left), Y_t is black and $y_t \geq a_t$. Lemma 13 applied to Y_t and X_t assures that there are neither free spaces nor white pebbles between the closing position of size $2x_t$ and j_t. However, since $y_t \geq a_t > x_t$ it follows from Lemma 10 applied to X_t and Y_t that A_t is black in Γ'_t. We argue that there are neither white pebbles nor free places between i_t and j_t: let l be the closing position of size $2x_t$. If $i_t \geq l$, the result follows; otherwise, since $a_t \geq 2x_t$ there are two possibilities: either there are some black pebbles of size $2x_t$ in Γ'_t – in this case it must hold that $a_t = 2x_t$ and due to Lemma 5 there is a continuous sequence of black pebbles of size $2x_t$ between i_t and l, or there are no black pebbles of size $2x_t$ in Γ'_t. However, in the latter case it cannot be that $i_t < l$. Hence, there is a continuous sequence of black pebbles between i_t and j_t in Γ'_t.

Consider now the colors of the pebbles in Γ'_{t+1}. A_{t+1} has size y_t and hence is black. All pebbles before i_t in Γ'_t (including possible pebble Q from line 10) retained their colors from Γ'_t, as did the pebbles after A_{t+1}. Pebble X_t starts at i_t and can be either black or white. Pebbles between X_t and A_{t+1} are black, because they were black in Γ'_t. Now we argue that the invariants Γ'_{t+1} is valid.

P1: First consider the situation when there was no swap on line 10. For pebbles before i_t the free bandwidth before them remains the same. The free bandwidth before any pebble after A_{t+1} in Γ'_{t+1} did not increase from Γ'_t: A_t was removed and A_{t+1} of size $y_t \geq a_t$ added. X_t was moved to i_t so the free bandwidth couldn't increase. The remaining pebbles in Γ'_{t+1} are the pebbles between X_t and A_{t+1} (including A_{t+1}). They form a continuous sequence of black pebbles of size at least a_t. Since the free bandwidth before X_t in Γ'_t was less than x_t, the free bandwidth before each of these pebbles in Γ'_{t+1} is less than $x_t + a_t - x_t$.

The possible swap on line 10 doesn't increase the free bandwidth before any pebble.

P2: Again, consider first the situation without a swap on line 10. The only possibility for two consecutive white pebbles in Γ'_{t+1} is that X_t has a white neighbor. However, X_t is followed by a sequence of black pebbles which is non-empty[7] the only way is that X_t has a left neighbor Q which is white, and hence preceded by a black pebble W. However, from Lemma 9 it follows that $q < x_t$, and from Lemma 10 it follows that $x_t \geq w$; hence X_t is black.

Now we argue that the swap on line 10 can not result in two consecutive white pebbles. That could only happen if there is a white pebble following X_t before the swap. However, X_t was the smallest pebble to the right of i_t, hence it's right neighbor must be black.

P3: Since X_t was white in Γ'_t, the pebble following A_{t+1} in Γ'_{t+1} is black. So the only way to violate **P3** in Γ'_{t+1} is by means of X_t. Suppose that X_t is white in Γ'_{t+1}. Then either A_t was the rightmost pebble[8] of size a_t in Γ'_t or right neighbor of A_t in Γ'_t is strictly bigger[9] than a_t. Hence, in order to violate **P3**, X_t must be black in Γ'_{t+1}, must have a white left neighbor Q which in turn has black left neighbor W of size x_t. Then, however, X_t is swapped with Q. The result follows by noting that Q has a black right neighbor of size at least $a_t \geq x_t > q$ (Y_t or some black pebble before it).

Case 2: $x_t \geq a_t$

First note that in this case, the swap on line 10 never happens. Indeed, if A_t was white in Γ'_t, it was immediately preceded by a black pebble which remains black also in Γ'_{t+1}. On the other hand, if A_t was black in Γ'_t and had a white left neighbor Q, then Q's left neighbor W was black, and $w < a_t \leq x_t$. Now consider the colors of the pebbles in Γ'_{t+1}. Pebbles before i_t have the same color

[7]If $A_t \neq Y_t$, then this sequence contains at least Y_t. If $A_t = Y_t$, then $a_{t+1} = a_t$ and since A_t is black in Γ'_t, A_{t+1} is black in Γ'_{t+1}.

[8]In case that $t = 1$ or $a_t = x_{t-1}$; see line 6 of procedure Delete.

[9]In this case $a_t = x_{t-1}$ and the claim follows directly from **P3** in Γ'_t.

as in Γ'_t. X_t can be either black or white, however, if it is white, then A_t must have been white in Γ'_t. A_{t+1} has the size x_t or y_t and is black. All other pebbles have the same color as in Γ'_t, because X_t was the smallest one. Let us proceed to show that the invariants hold in Γ'_{t+1}:

P1: Clearly, the free bandwidth before pebbles to the left of i_t remains the same. Since $a_{t+1} \geq x_t \geq a_t$, the invariant holds for pebbles to the right of X_t. Finally, the invariant holds for X_t.

P2: Because A_{t+1} is black, the only possibility for two consecutive white pebbles in Γ'_{t+1} is that X_t is white and has a white neighbor. However, if X_t is white in Γ'_{t+1} then A_t was white in Γ'_t, and had two black neighbors. Let L be A_t's left neighbor from Γ'_t. Since X_t is white in Γ'_{t+1}, due to Lemma 9 it holds $a_t < x_t < l$. If the right neighbor of A_t in Γ'_t was different from X_t, it was black and remains black in Γ'_{t+1}. Hence, in order to violate **P2**, let the right neighbor of A_t in Γ'_t be X_t. Since $x_t < l$, it is a contradiction with the fact that Γ'_t was valid.

P3: Since Γ'_t was valid, there are only two places where the **P3** could be violated in Γ'_{t+1}: the violating configuration must involve either X_t or A_{t+1}. Since A_{t+1} is black in Γ'_{t+1}, either X_t was black in Γ'_t and nothing changed, or X_t was white and preceded by a black pebble of size x_{t+1}, and the invariant holds, too.

Let us suppose that the violating configuration contains X_t. If X_t is white in Γ'_{t+1}, then A_t was white in Γ'_t and **P3** holds. If X_t is black in Γ'_{t+1}, the violating configuration must involve a white neighbor Q of X_t. However, if Q is X_t's right neighbor, then $x_t = a_t$, and **P3** holds. Hence, Q is X_t's left neighbor, and following the argument at the beginning of Case 2, **P3** cannot be violated.

Now that we have proved the claim, let us consider the last iteration of the algorithm. At the beginning, the statement of the claim holds. We shall argue that at the end of the algorithm invariants **P1**–**P3** hold. Let Γ_t be the final configuration.

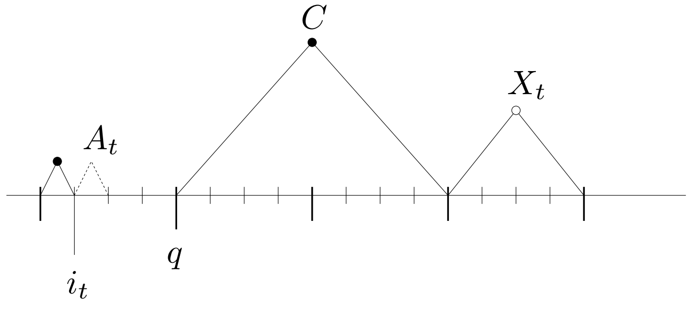


Figure 14: A possible situation in the last iteration of the **while** loop.

There are two reasons for the algorithm to stop. One of them is if, at the beginning of an iteration, there are no pebbles to the right of i_t. Then Γ_t is obtained from Γ'_t by deleting the rightmost pebble A_t so all invariants hold. The other possibility to stop the algorithm is when the selected pebble X_t does not fit to position i_t. Let C be the first pebble after i_t in Γ_t (see Figure 14). As $x_t \geq 2a_t$ we can argue that there is no free place between C and X_t in Γ'_t (and hence in Γ_t). Indeed, if X_t is black in Γ'_t, it holds $x_t \geq c$, but $x_t \leq c$ since X_t was the smallest pebble. Hence $x_t = c$ and due to Lemma 5 there is no free place between C and X_t. On the other hand, if X_t is white in Γ'_t, the claim holds due to Lemma 13.

Moreover, the colors of the pebbles in Γ_t and Γ'_t are the same, as can be seen by considering the removal of A_t from Γ'_t. Pebbles to the left of A_t don't change color. Pebbles after A_t are all bigger than A_t (X_t was smallest of them and did not fit into A_t's place), so deleting a smaller pebble before them cannot change their color. Now let us prove that the invariants hold in Γ_t.

P1: Let q be the position of C; we shall argue that $q - i_t \leq x_t - a_t$. Obviously, q is a multiple of x_t. Since Γ'_t is valid, A_t is fully contained in the place of size x_t ending at q. Now suppose for the sake of contradiction that $q - i_t > x_t - a_t$. Since i_t is a multiple of a_t, it must hold that $q - i_t \geq x_t$. But then, X_t would fit at i_t – a contradiction.

The free bandwidth before i_t in both Γ_t and Γ'_t is at most $a_t - 1$. From the previous claim it follows that the free bandwidth before any pebble located after i_j in Γ_t is at most $a_t - 1 + x_t - a_t = x_t - 1$. Since all the considered pebbles are of size at least x_t, the invariant holds.

P2: Since C is bigger than A_t and Γ'_t is valid, C is black in both Γ_t and Γ'_t. Hence, the invariant could not be violated by deleting A_t.

P3: Since C is black, the only way to violate **P3** is if A_t's left neighbor, W, is white and has black left neighbor of size c. However, then both W and A_t would have been white in Γ'_t – a contradiction.

□

As a last part in this appendix, we present the full proof of Lemma 3 concerning the complexity of DELETE:

Proof of Lemma 3 First, we treat a special case when no iterations were performed, i.e. the rightmost pebble A of given size a, starting at i, was removed, and either there are no pebbles to the right of A, or the smallest such pebble X does not fit at i. In this case the situation is valid due to Theorem 2, and what remains is to show how to maintain **P4**. If A was the rightmost pebble, the free place formed by removing A have no pebbles to the right. For all other free places, the size of the smallest pebble to their right could not decrease, from which it follows that **P4** holds. If X does not fit at i, it must be that all pebbles to the right of A are bigger than A. Hence, it is sufficient to put one coin at A, and using similar arguments we get that **P4** holds.

For the rest of the proof suppose that there is at least one full iteration of the main loop. Recall the notation from the proof of Theorem 2, i.e. we number the iterations of the main loop, and Γ_t is the configuration at the beginning of

t-th iteration. Moreover, Γ'_t is obtained from Γ_t by purring a new pebble A_t on P_t. From the proof of the theorem it follows that Γ'_t is always a valid situation. We prove by induction on t that the following can be maintained:

For every iteration $t > 1$ of the **while** *loop, the corresponding Γ'_t satisfies* **P1**–**P4**, *all pebbles to the right of i_t have size at least a_t, and one extra coin lays on A_t. Moreover, if some pebble to the right of i_t has the size a_t, then two extra coins lay on A_t.*

Basis: Consider Γ'_2. It is obtained from the initial situation (Γ'_1) by removing $A_1 = A$, moving X_1 to i_1 with an optional swap with its left neighbor, and placing A_2 on $i_2 = j_1$. We prove that all pebbles to the right of i_2 have size at least a_2. X_1 was selected as the smallest and rightmost pebble to the right of i_1, hence all pebbles to the right of j_1 are bigger than x_1. If X_1 was black in Γ_1, $a_2 = x_1$, so let us suppose that X_1 was white, and $a_2 = y_1$ (see Figure 6). Then by Lemma 10 all pebbles to the right of j_1 are of size at least y_1.

Let C be A_1's right neighbor in Γ_1, and q be its starting position (see Figure 15). Note that there is no free bandwidth between q and j_1 in Γ_1: Since X_1 is the smallest pebble to the right of i_1, $c \geq x_1$, and q is a multiple of x_1, and so are the sizes of all pebbles between C and X_1. Obviously, there is no free place between C and X_1, because otherwise there would be at least x_1 free bandwidth before X_1 – a contradiction with **P1**.

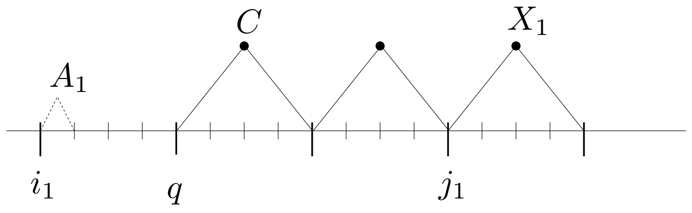


Figure 15: There is no free bandwidth between q and j_1.

What remains to be proven is that it is possible to maintain **P4** in Γ'_2; this is sufficient because the one or two extra coins to be put of A_2, and the coin needed for the first iteration can be both charged to the constant number of coins allowed for the DELETE.
Let us now distinguish two cases:

Case 1: $x_1 \geq a_1$. In this case the swap on line 10 cannot happen (see Case 2 in the proof of Theorem 2), so Γ'_2 is obtained by removing A_1, moving X_1 to i_1, and placing A_2 on j_1. Obviously, for all places to the left of i_1, **P4** remains valid, since the smallest pebble to the right could not decrease. Also, places to the right of j_1 were not affected. Hence, we can restrict ourselves to places between i_1 and j_1. However, there is no free place between i_1 and j_1 in Γ'_2: there is no free place between q and j_1 in Γ'_1 (and thus also in Γ'_2). Moreover, since $a_1 \leq x_1$, i_1 is a multiple of x_1 (X_1 fits at i_1), and due to **P1**, it holds that $q - i_1 = x_1$, and there is no free place between i_1 and j_1 in Γ'_2.

Case 2: $x_1 < a_1$. First, consider the case without the swap on line 10. As above, it is sufficient to prove that **P4** holds for free places between i_1 and j_1.

Note that in Γ'_1 there is no free place between i_1 and j_1: if there would be a free place, it would have to be between A_1 and C, which would, in turn, lead to contradiction with **P1** applied to X_1. Hence, the only free places of interest are those remained from P_1 after placing X_1 on i_1.

Since $x_1 < a_1$, X_1 is white in Γ'_1, and $a_2 = y_1 \geq a_1 > x_1$. We prove that all pebbles to the right of i_1 in Γ'_2 (except X_1) are of size at least a_1. We already know that all pebbles to the right of i_2 have size at least $a_2 \geq a_1$. Since X_1 was selected as the smallest one, all pebbles between i_1 and j_1 in Γ'_1 are of size at least x_1. However, due to Lemma 9 they must be black, and thus of size at least a_1. The same pebbles are in Γ'_2.

Consider any free place P in Γ'_2 created after the removal of A_1. Since the smallest pebble to the right of P is of size at least a_1, if $p < a_1/2$, no coins need to be placed on P. However, $p \leq a_1/2$, and there is at most one P for which $p = a_1/2$. Hence, one coin placed on P is sufficient to maintain **P4**.

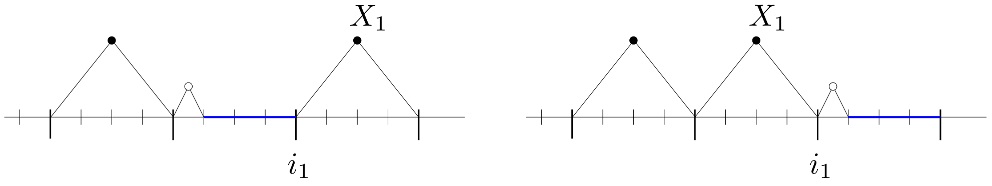


Figure 16: Moving free places during swap.

The last thing to note is how to handle the swap on line 10. As can be seen from Figure 16, the only free places that were affected by the swap are those immediately to the left of i_1. However, the coins may be moved to corresponding places; obviously, the smallest pebble to the right cannot decrease, so we conclude that the new allocation of coins satisfies **P4**.

Induction step: In order to prove the claim for Γ'_{t+1}, consider the situation when the algorithm finishes the t-th iteration and enters the $t+1$st. Γ'_{t+1} is obtained from Γ'_t by removing A_t, moving X_t to i_t, and placing A_{t+1} on $i_{t+1} = j_t$. Note that in this case $x_t \geq a_t$, so the is no swap. Using the same arguments as above we argue that all pebbles to the right of i_{t+1} have size at least a_{t+1}, and that there is no free place between i_t and j_t in Γ'_{t+1}. Since for the free places before i_t and after j_t, **P4** remains valid, we argue that **P4** folds in Γ'_{t+1}.

Now we show how to find two free coins – one to pay for this iteration, and one to be placed on A_{t+1}. If $x_t = a_t$, there are two coins placed on A_t. Otherwise (i.e. in case that $x_t > a_t$) one coin comes from the deletion of A_t and the other can be found as follows. Since X_t was placed on i_t, and $x_t > a_t$, there must have been a free place P of size $x_t/2$ in Γ'_t. Moreover, X_t was to the right of P, and so there must have been at least one coin on P. In Γ'_{t+1}, P is covered by X_t, so the coin can be used.

The last thing to show is to find a second free coin in case that there exists some pebble Q to the right of A_{t+1} such that $q = a_{t+1}$. In this case X_t is white in Γ'_t – otherwise it would hold that $a_{t+1} = x_t$ and there would be no pebble of size x_t to the right of X_t. Hence $x_t \leq a_{t+1}/2$ and A_{t+1} in Γ'_{t+1} covers a place of size $a_{t+1}/2$ that has been free in Γ'_t. According to **P4** there is a coin on this place in Γ'_t; this coin can be used.

The proof of the theorem is concluded by considering the last iteration. Let $\Gamma'_{t_{fin}}$ be the last situation. The final situation is obtained from $\Gamma'_{t_{fin}}$ by removing $A_{t_{fin}}$. We show that **P4** holds. The only free place that could violate **P4** is the one remained after $A_{t_{fin}}$, however, there was a coin on $A_{t_{fin}}$, and all pebbles to the right (if any) are bigger that $a_{t_{fin}}$.

□